The Need for Well-Factored Dynamic Parallel Programming Systems, and Why Charm++ is a Good Choice

Phil Miller

Advanced Modeling & Simulation (AMS) Seminar Series,
NASA Ames Research Center, September 12, 2016

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Charmworks

PARALLEL PROGRAMMING LAB
DEPT. OF COMPUTER SCIENCE, UNIVERSITY OF ILLINOIS
PPL UIUC

(Over)Decomposition

- Decompose the work units & data units into many more pieces than execution units
  - Cores/Nodes/..
- Not so hard: we do (over) decomposition anyway
  - Multi-block and AMR on structured meshes
  - Unstructured mesh partitions
  - N-body tree code ‘leaves’
  - Blocked linear algebra

Task/Object Placement

- Those pieces have to go somewhere
- Naïve reasons:
  - Matching (range of) indices
  - Easy to compute ID<->place mapping
- Good reasons: (e.g.)
  - Load balance
  - Communication locality on core/node
  - Communication locality over network
  - Workload affinity for hardware type

Task/Object Movement

- Is the chosen placement the best possible?
- Can you change it?
  - Without restarting the application?
- Why?
  - Load not as predicted (or assumed)
  - Load changed
  - Hardware performance changed(?!)
  - Part of hardware failed

Asynchronous Execution

- Work shouldn't have to 'wait its turn'
- Components should be willing to share
- I.e., Composition of independent tasks
  - Steps of a parallel algorithm
  - Multi-module and multi-physics
  - Using all hardware resources, all the time

Missing Optimizations

- Second-order placement effects
- Load balancing frequency
- Dynamic critical paths
- Energy usage
- **Productivity**

Why do it all by yourself, in every app?

Instantiations of (some of) these ideas

- Charm++
  - Including Adaptive MPI
- KAAPI
- HPX
- StarPU
- OmpSs
- ParSEC
- CnC
- Chapel
- X10
- Every AMR framework
  - Especially Uintah
- ProActive
- FG-MPI
- MPC

Why Charm++

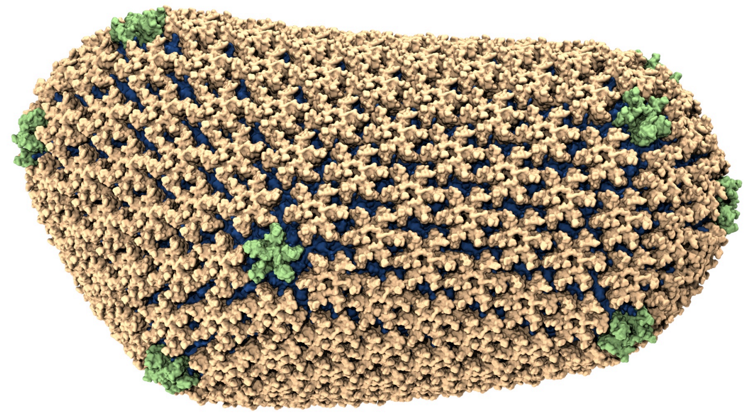
- Application experience
  - NAMD, EpiSimdemics, ChaNGa
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

Why Charm++

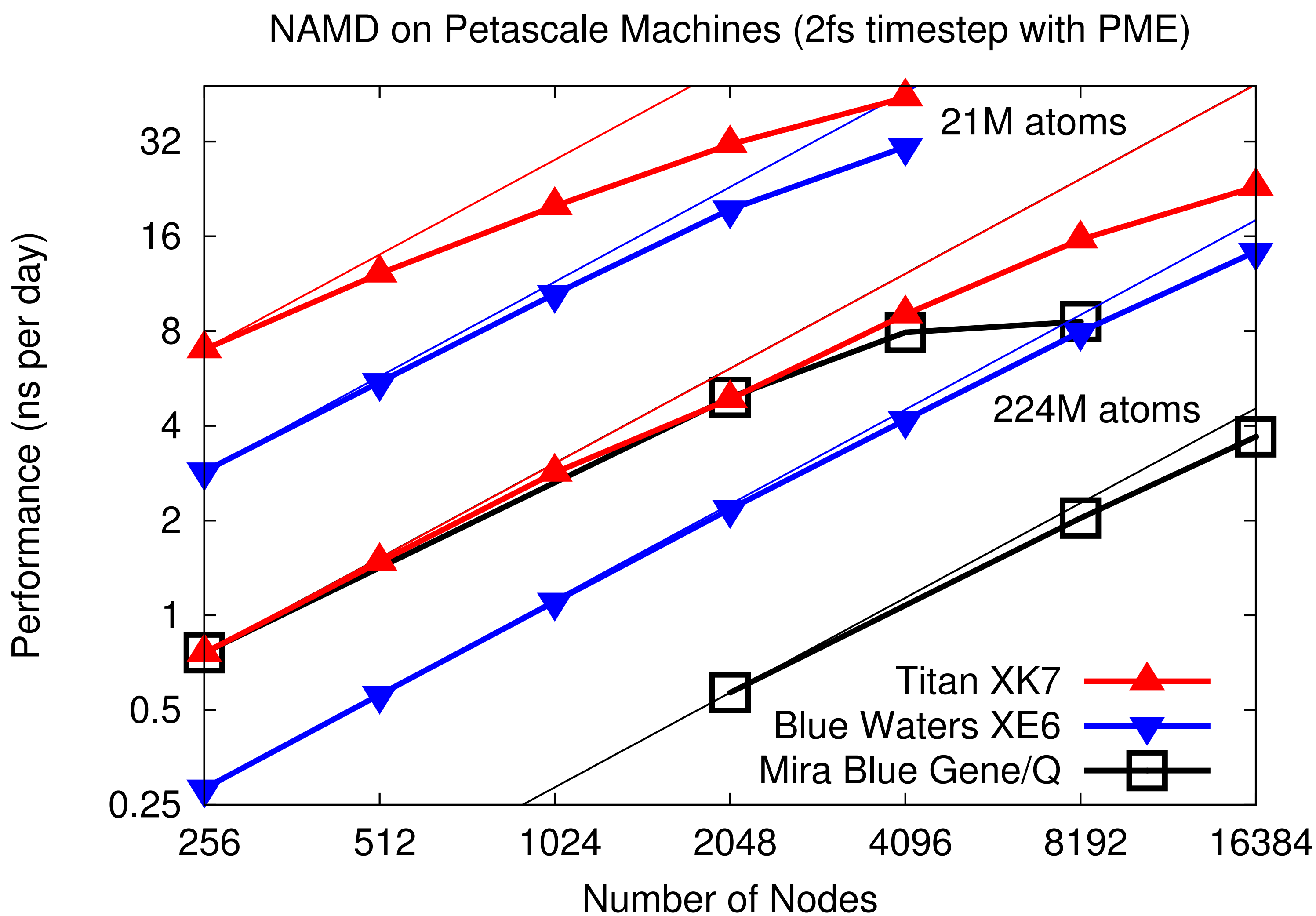
- Application experience
  - NAMD, EpiSimdemics, ChaNGa
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

NAMD: Biomolecular Simulations

- Long-term collaboration (1994-now) with K. Schulten
- Over 70,000 users
- Scaled to top US supercomputers
- 2002 Gordon Bell award
- 2012 Fernbach award



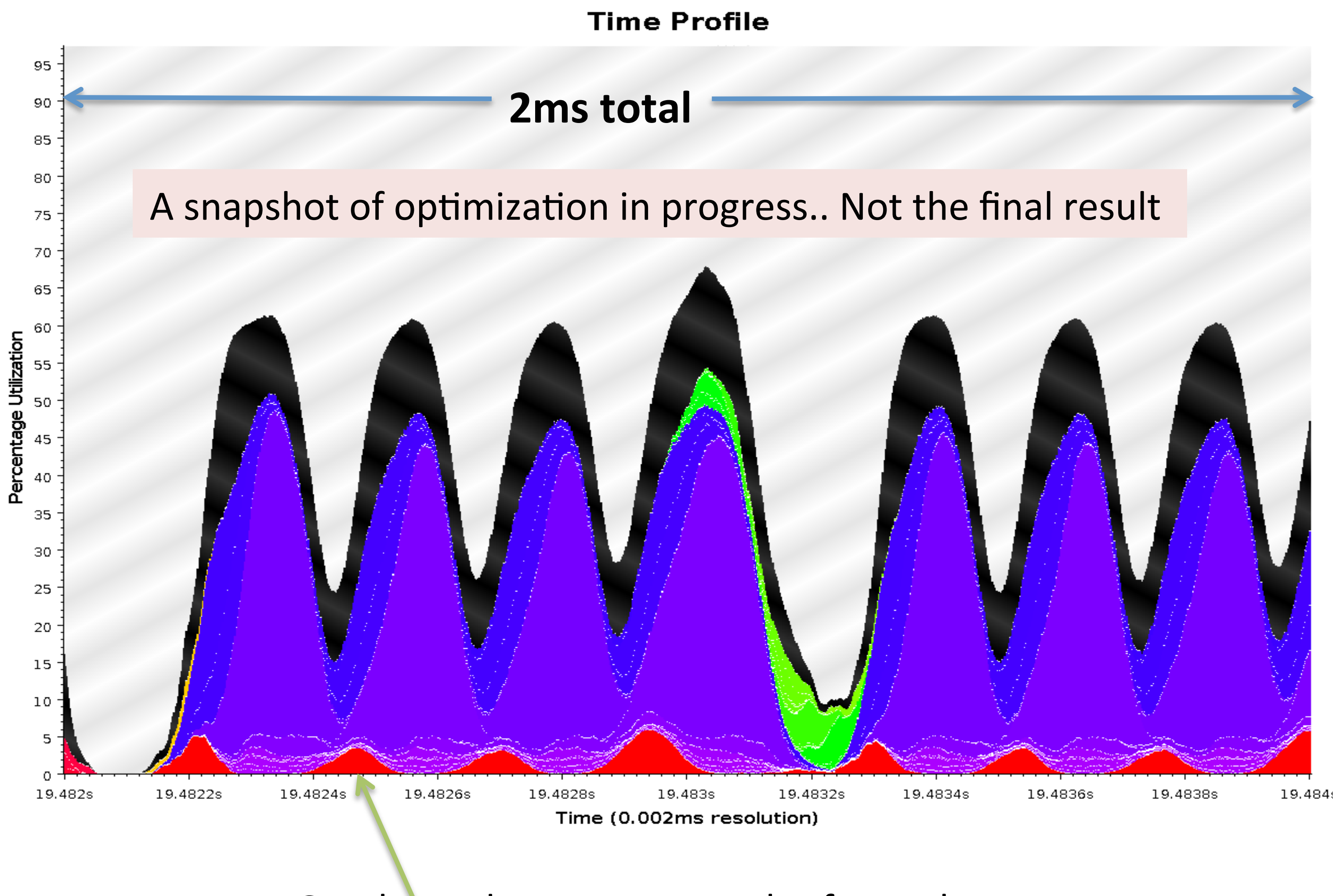
Science cover article:
Determination of HIV capsid structure



NAMD strong scaling on Titan Cray XK7, Blue Waters Cray XE6, and Mira IBM Blue Gene/Q for 21M and 224M atom benchmarks

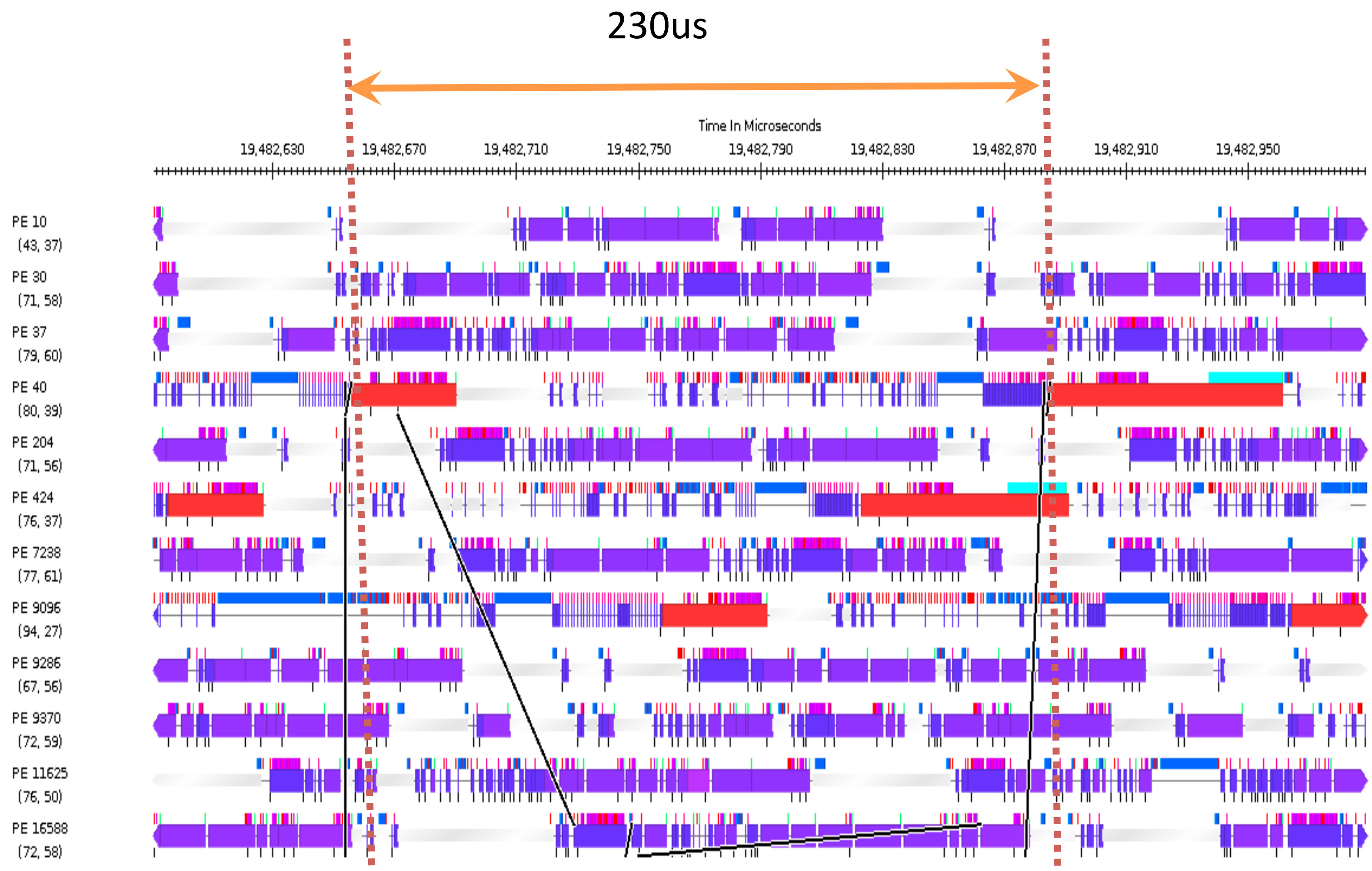
Time Profile of ApoA1 on Power7 PERCS

92,000 atom system, on 500+ nodes (16k cores)



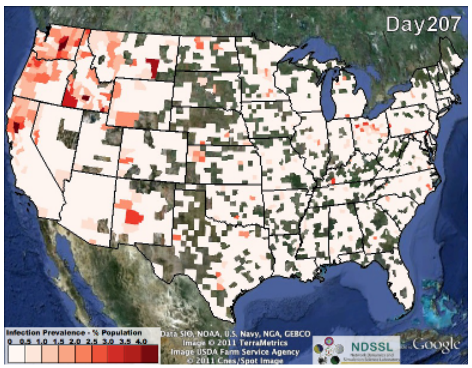
Overlapped steps, as a result of asynchrony

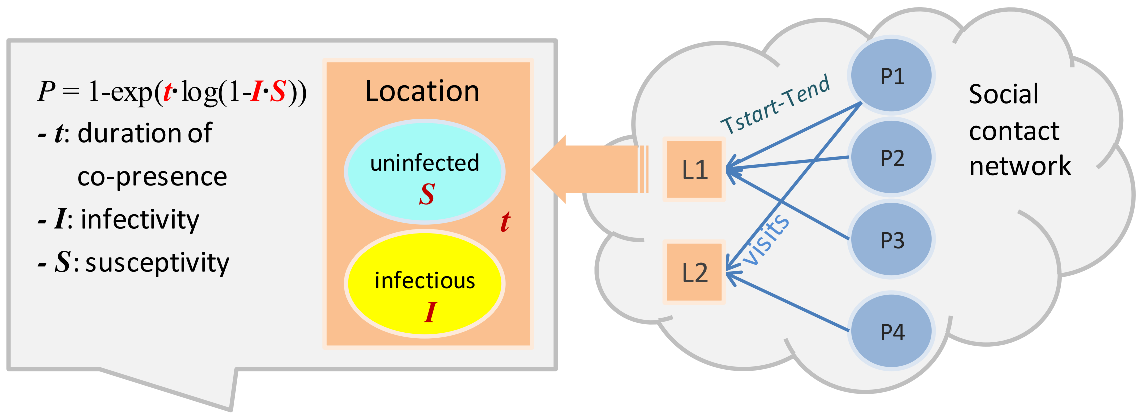
Timeline of ApoA1 on POWER7 PERCS



EpiSimdemics

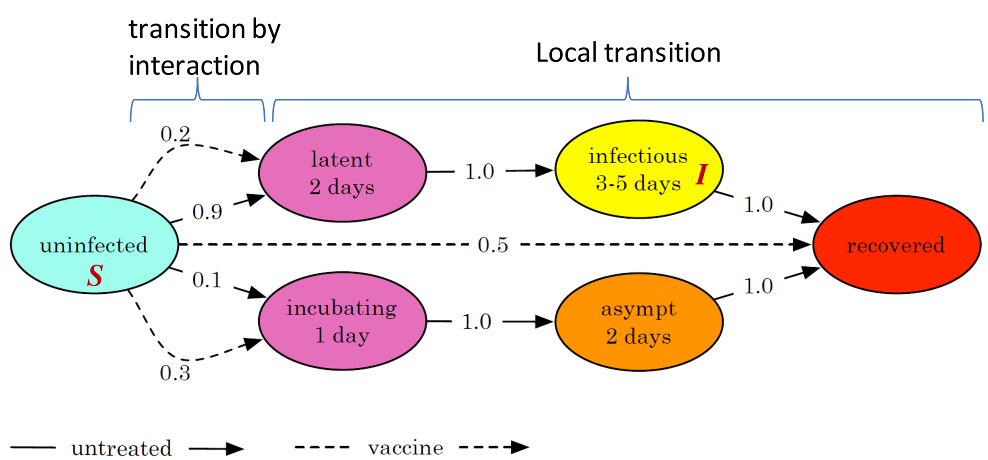
Simulating contagion over dynamic networks





EpiSimdemics[1]

- Agent-based
- Realistic population data
- Intervention[2]
- Co-evolving network, behavior and policy[2]

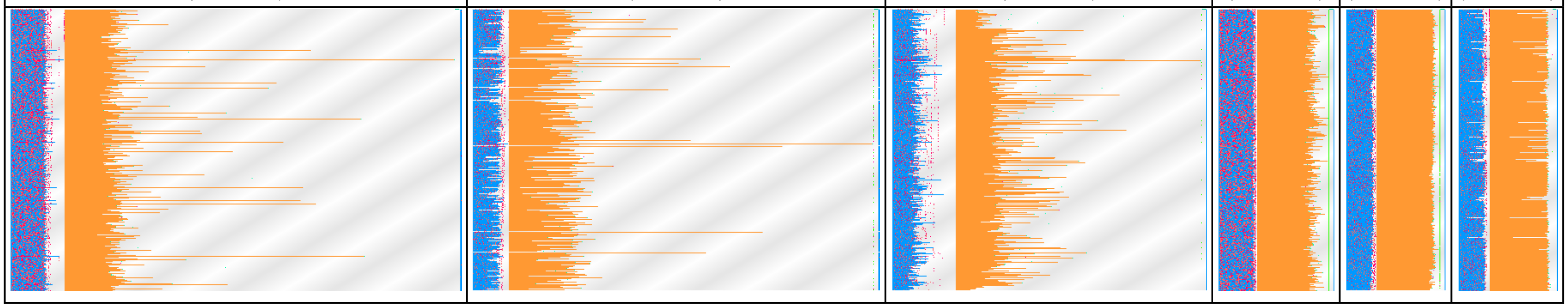


Conversion to Charm++

- Original in MPI, scaling failed at 512 cores
- Headline features:
  – Asynchronous reductions
  – Easy decomposition experiments
  – Streaming all-to-all
  – Composition - split mid-run for multiple scenarios, overlap on full job partition

Load distribution (Vulcan)

RR (1.755 s)	GP (1.583 s)	Z splitLoc (1.222 s)	RR splitLoc (0.438 s)	ZC splitLoc (0.369 s)	GP splitLoc (0.368 s)



splitLoc: **no peak in location computation**
Z-splitLoc: **no load balance**

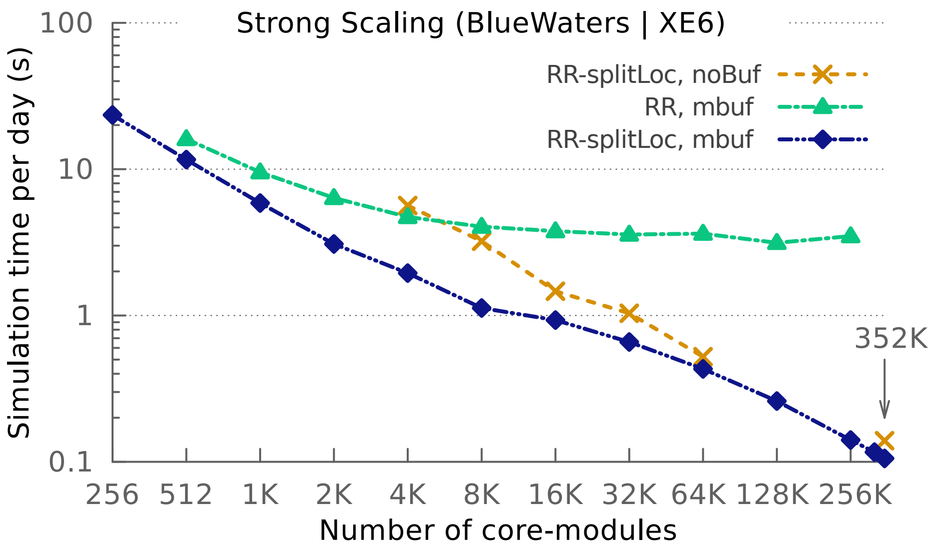
GP: **shorter person phase**
ZC-splitLoc: similar perf. w/ GP-splitLoc

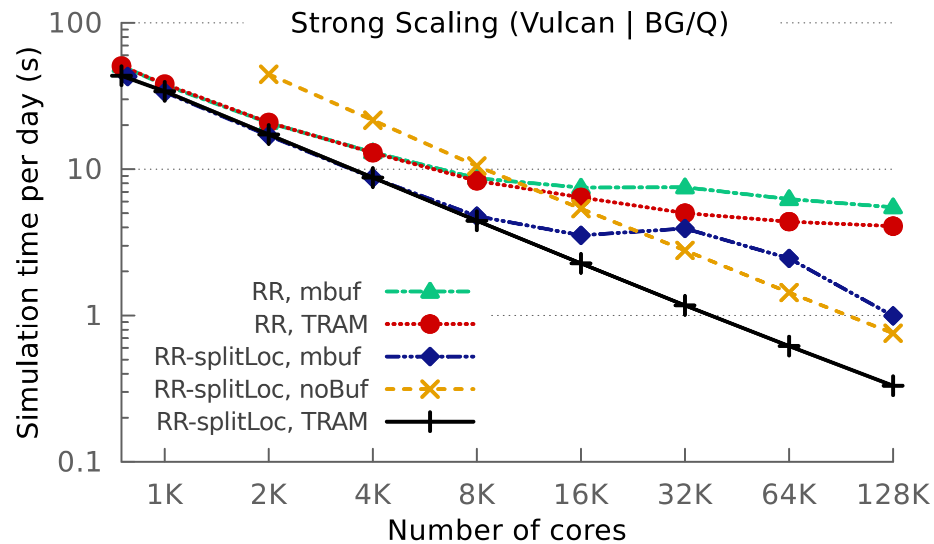
- Blue: person computation
- Red: receiver's msg handling
- Orange: location computation

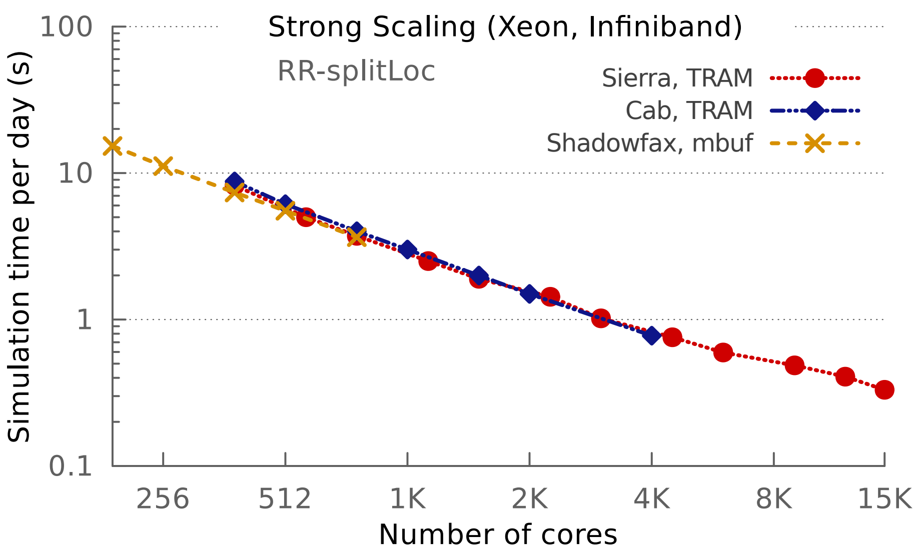
X-axis: **Time** Y-axis: **Processor**

Timeline of an iteration from sampled subset of 332 cores of total 4K using Michigan data on Vulcan

Strong scaling performance with the largest data set

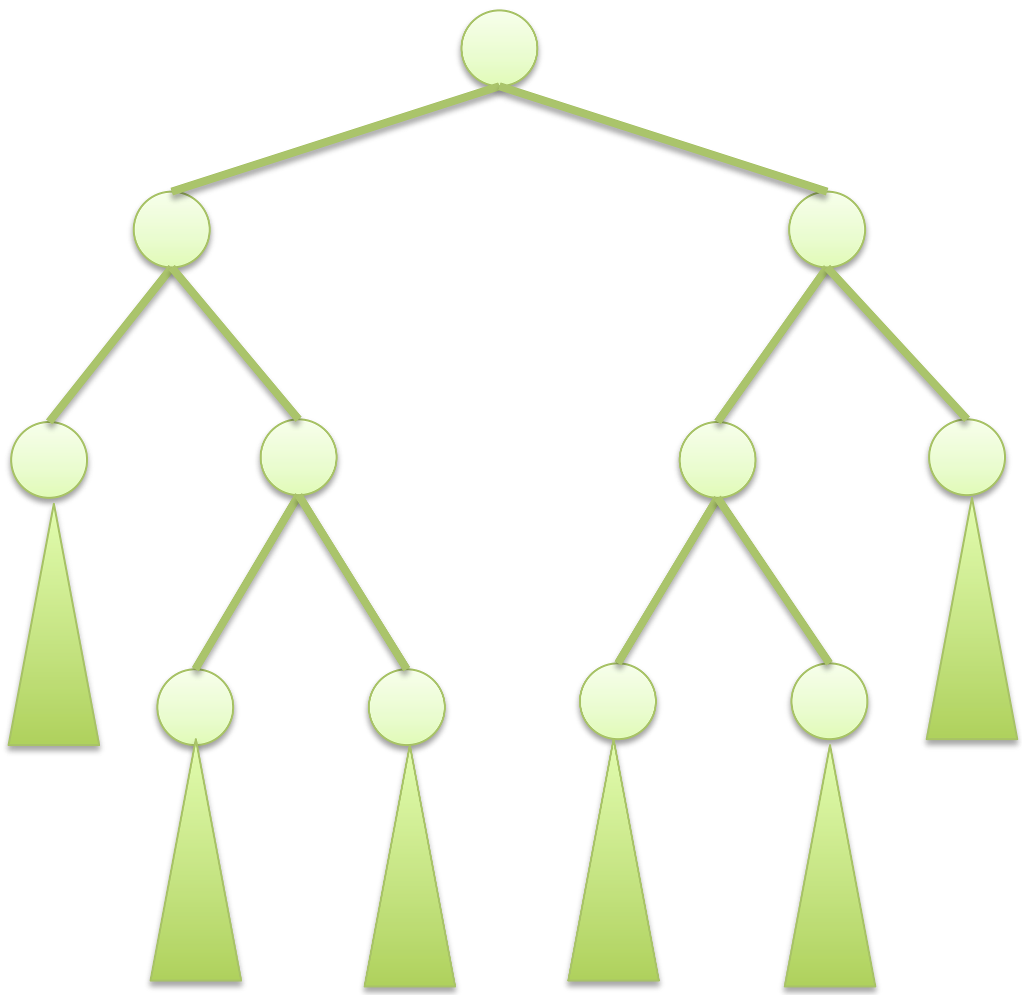






- Contiguous US population data
- **XE6**: **the largest scale** (**352K cores**)
- **BG/Q**: good scaling up to 128K cores
- Strong scaling helps timely reaction to pandemic

ChaNGa: Cosmology Simulation



Collaboration with
Tom Quinn at UW

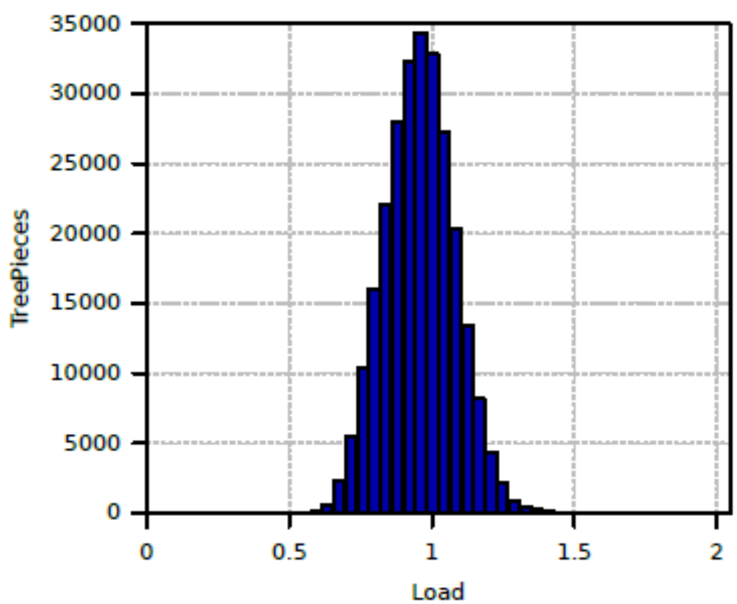
- Tree: Represents particle distribution
- TreePiece: object/ chares containing particles

Multiple time-stepping!

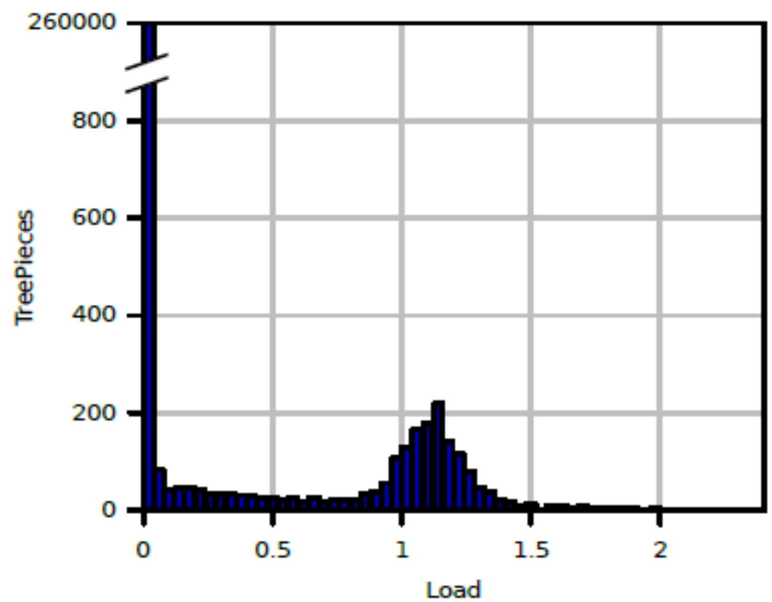
- Our scientist collaborators suggest an algorithmic optimization:
  - Don't move slow-moving particles every step
    - i.e. don't calculate forces on them either
  - In fact, make many (say 5) categories (rungs) of particles based on their velocities
  - Rung sequence (with 5 rungs)
    - 4 3 4 2 4 3 4 1 4 3 4 2 4 3 4 0
    - Rung 0: all particles, Rung 4: fastest-moving particles
  - Each tree-piece object now presents a different load when different "rungs" are being calculated

Multiple time-stepping!

- Load (for the same object) changes across rungs
  - Yet, there is persistence within the same rung!
  - So, specialized phase-aware balancers were developed



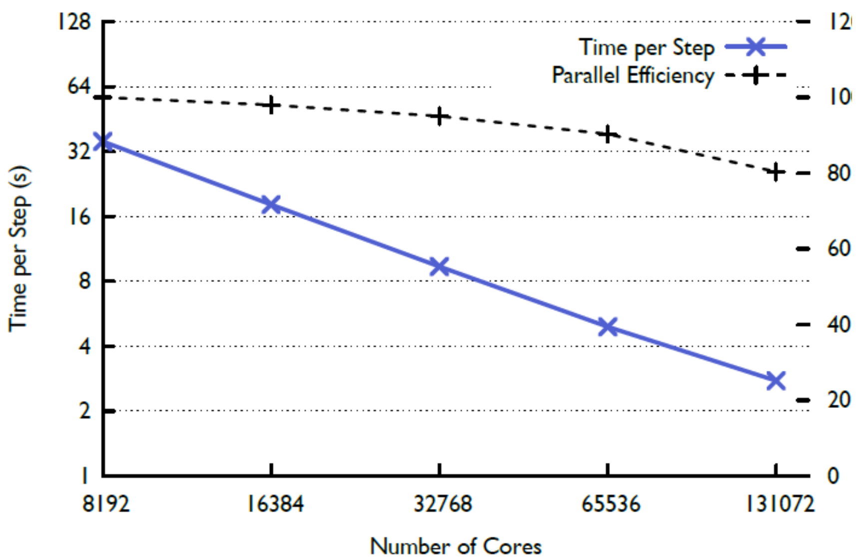
(a) Rung 0



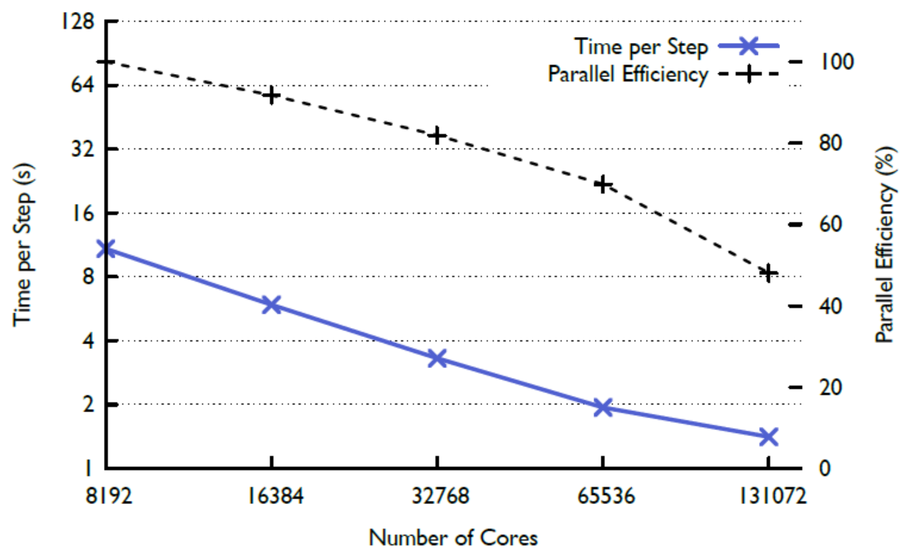
(b) Rung 4

Multi-stepping tradeoff

- Parallel efficiency is lower,
  but performance is *much better*



Single Stepping



Multi Stepping

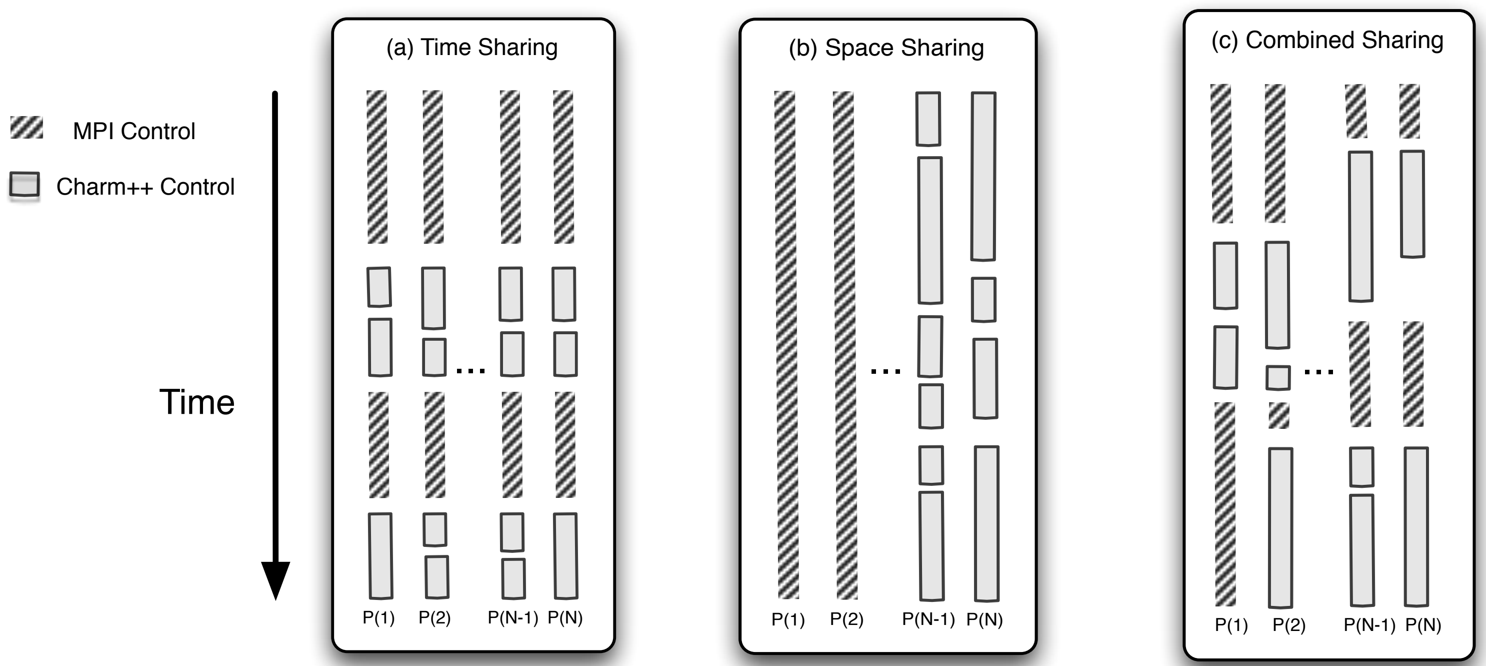
Why Charm++

- Application experience
  - NAMD, EpiSimdemics, ChaNGa
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

MPI Interoperability

- Make calls between MPI and Charm++ code
- Implement each parallel kernel in the most suitable model
- Code shares process address space
  - Can pass plain pointers across interface
- Control transfer between Charm++ and MPI analogous to MPI code calling external libraries (e.g. ParMETIS, FFTW, PETSc, Hypre)

MPI Interoperability Modes



MPI Interoperability Code

- Include `mpi-interoperate.h`
- Add an interface function callable from the main program

```
void HelloStart(int elems)
  if(CkMyPe() == 0) {
    CProxy_MainHello mainhello =
    CProxy_MainHello::ckNew(elems);
  }
  StartCharmScheduler();
}
```

MPI Interoperabilty: Control Flow

- Begin execution at user main
- Perform MPI initialization and application initialization
- Create a communicator for Charm++
- Initialize Charm++
- for (as many times needed)
  - perform MPI based communication and application work
  - invoke Charm++ code
- Exit Charm++
- Exit MPI

MPI Interoperability: Example Code

```
MPI_Init(argc,argv); //initialize MPI
//Do MPI related work here
```

```
//create comm to be used by Charm++
MPI_Comm_split(MPI_COMM_WORLD, myRank % 2, myRank, newComm);
CharmLibInit(newComm,.) //initialize Charm++ over my communicator
```

```
if(myRank % 2)
  StartHello(); //invoke Charm++ library on one set
else
  //do MPI work on other set
```

```
kNeighbor(); //invoke Charm++ library on both sets
CharmLibExit(); //destroy Charm++
```

MPI Interoperability: Use Cases

- Demonstrated in HPC Challenge submission with FFT benchmark
- Chombo AMR framework using parallel sorting library from
  - Highly Scalable Parallel Sorting by Edgar Solomonik and Laxmikant Kale (IPDPS, 2009)
- EpiSimdemics using MPI-IO

Why Charm++

- Application experience
  - NAMD, EpiSimdemics, ChaNGa
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

Development: Portability

- Compilers: GNU, Intel, IBM, Clang, Cray, PGI
- Network: BlueGene *, Cray *, IB Verbs, TCP/IP
- CPU Architectures: x86, POWER, BG *, ARM
- OS: Linux, Mac, Windows, BG *

Development: Stability

- Nightly cross-platform testing
- Thorough test coverage
- Continuous Integration against applications
- Code Review of every commit
- RTS runs clean under Valgrind, ASan, & UBSan
- SMP build is mostly ThreadSanitizer clean

Development: Compatibility

- Frivolous API changes avoided
- NAMD always tested for compatibility, forward and backward

Why Charm++

- Application experience
  - NAMD, EpiSimdemics, ChaNGa
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

Features & Ecosystem

- Automatic offline & online fault tolerance
  - Checkpoint in one line, transparent restart, **any number of processors**
  - Need platforms, vendors to support resilient jobs!
- Plethora of LB strategies
  - Easy to plug in your own
- Scalable tools
  - CharmDebug parallel debugger
  - LiveViz online visualization client
  - Projections performance analysis tool
- Resource Optimizations
  - In-job power & energy: need freedom to control DVFS/RAPL
  - Job size tuning via shrink/expand: need cooperative scheduler
  - Across-job power & energy: scheduling with power constraints

Why Charm++

- Application experience
  - NAMD, ChaNGa, EpiSimdemics
  - OpenAtom, Fractography, Stochastic MIP, Cloth Simulation
- Interoperation with native MPI code
  - Easy, low risk, incremental adoption
- Production-ready development
  - Portability
  - Stability
  - Compatibility
- Rich, Extensible Ecosystem
- Comprehensive feature set

Questions?